ОБОБЩЕНИЯ И ОПТИМИЗАЦИИ НА НЯКОИ АЛГОРИТМИ[1]

Антон Илиев, Никола Вълчанов, Тодорка Терзиева

**4003 Пловдив, бул. „България” 236,*
Пловдивски университет „Паисий Хилендарски”
Факултет по математика и информатика,
***1113 София, ул. Акад. Георги Бончев, бл. 8,*
Институт по математика и информатика,
Българска академия на науките
e-mail: aii@uni-plovdiv.bg, nvalchanov@gmail.com, dora@uni-plovdiv.bg

В тази статия са представени техники за обобщение и оптимизация на някои алгоритми от теория на числата и един от аналитична геометрия, намиращи практическо приложение в други задачи. Всички алгоритми имат софтуерна реализация в Microsoft Visual Studio 6.0 SP 6 като конзолни C++ приложения.

Ключови думи: *алгоритми, обобщение и оптимизация на алгоритми.*

I. Търсене на броя на простите числа в даден интервал.

Идея на метода в общия случай: Означаваме с $n_1, n_2, \ldots, n_k$ първите k прости числа, подредени в нарастващ ред. След това се намират всички числа, които са по-малки от $n_1 n_2 \ldots n_k$ и не се делят на $n_1, n_2, \ldots, n_k$. За по-голяма яснота всички числа с това свойство ще наричаме числа със свойство A. Разликите между намерените числа със свойство A ни дават стъпките, а търсенето на следващи кандидати за прости числа започва от следващите прости числа n_{k+1} и n_{k+2}, които са първите по-големи от n_k ($n_k < n_{k+1} < n_{k+2}$). Числото P е такова, че поставено на последно място в редицата от стъпки (получена от разликите на числата със свойство A), прави така че сумата на елементите на тази редица да стане точно равна на $n_1 n_2 \ldots n_k$. След това намираме броя на стъпките и по този начин можем да прескачаме всички кратни на $n_1, n_2, \ldots, n_k$ числа. Броят на елементите в поредицата от стъпки, получена от разликите между намерените числа със свойство A е равен на

(1) $$(n_1 - 1)(n_2 - 1)\ldots(n_k - 1).$$

Програмна реализация: С променливата par е означено $k-1$. Приложената програмна реализация работи за случаите par= 0, 1, 2, 3, 4, 5 и 6 (в соурс кода в момента par=6). Първо се „отсяват” само числата, които не се делят на $n_1, n_2, \ldots, n_{\text{par}+1}$, а от разликите между тях се получава поредицата от стъпки. Лесно може да се пресметне по формула (1), че за par=6 имаме нужда от 92160=1*2*4*6*10*12*16 на брой стъпки и поради това сме преценили, че за практическо приложение на метода това е максимумът. Имат значение не само намерените стъпки, а и от коя стъпка трябва да се започне, за да може алгоритъма да се обобщи и да бъдат нещата коректни. В програмната реализация сме решили и този проблем. Например, ако par=6, то *(r+92160)=4, i=1, където i=1 дефинира индекса на елемента в масива от стъпки, от който да се започне, *(r+92160) е гореспоменатото число P. След това се прилага оптимизацията, че ако едно естествено число S не се дели на никое просто число (започвайки от 19 включително нагоре, тъй като поради подходящо избраната поредица от стъпки прескачаме всички кратни на 2, 3, 5, 7, 11, 13 и 17 числа при par=6) по-малко или равно на $\sqrt{S}$, то S е просто. Разглеждания подход обобщава известните в литературата подходи поради предварителното определяне на по-голяма стъпка и по-малкия брой деления при проверка дали едно число е просто. Като частен случай на разглеждания подход се получават алгоритмите споменати в книгите [1] par= 2, [2] par= 0, 1, 3, и други. В програмата се търси броя на простите числа по-малки от 1000000000.

```
#include<iostream.h>
#include<math.h>
void main()
{ unsigned long w[6543],j,u=0,p,k,i,r[92168],par=6,pr,x,t,q,ro,gg=1000000000;
  bool g;
   *w=2;*(w+1)=3;*(w+2)=5;*(w+3)=7;*(w+4)=11;*(w+5)=13;*(w+6)=17;
```

[1] Тази работа е финансирана по проект ИСМ-4 към поделение „Научна и приложна дейност” на ПУ „Паисий Хилендарски”.

```
  *(w+7)=*(r+7)=19;k=7;j=23;p=par+1;pr=2;for(t=1;t<=par;t++) pr*=*(w+t);
  while (u<pr)  {       g=true;t=0;
                do if (j%*(w+ ++t)==0) {g=false;break;} while (t<par);
                if (g) {ro=*(r+k-p)=j-*(r+k);u+=ro;*(r+ ++k)=j;}
                j+=2;       }
 if (par>0) {x=++k-par-2;q=7-par;*(r+x)=4;} else {x=0;q=0;*r=2;}  k=7;x++;j=23;i=q;
 while (j<gg)  {        p=unsigned long(sqrt(j));g=true;t=par;
                do if (j%*(w+ ++t)==0) {g=false;break;} while(*(w+t)<p);
                if (g) if (++k<6543) *(w+k)=j;   j+=*(r+i++); if (i==x) i=q;       }
              cout<<++k<<"->"<<gg<<endl;
}
```

Резултат от програмата: 50847534.

II. Търсене на броя на всички прости делители (всеки прост делител се брои колкото пъти се среща, отчитат се всички срещания) на числата от интервала [2; N].

Идея на метода в общия случай: Първо се намират всички прости числа, които са по-малки или равни на $\sqrt{N}$. Използва се следната оптимизация [2]: кандидатите за прости числа, които са по-големи или равни на пет са от вида $6k \pm 1,\ k = 1, 2, 3, \ldots$. По този начин се прескачат числата кратни на 2 и 3.

След това се използват фактите, че: едно естествено число винаги може да се представи като произведение само на прости делители, някои, от които могат да бъдат и повече от един път срещащи се, отчитат се всички срещания; ако едно естествено число P няма делители, които са по-малки или равни на $\sqrt{P}$, то P е просто и следователно процеса на търсене на делители трябва да спре, защото такива няма да има.

Програмна реализация: Първата стъпка е да намерим всички прости числа от интервала [2; Q], където Q е първото просто число по-голямо от $\sqrt{2^{32}-1}$. За Q използваме, че е 6543-тото поред просто число. След което търсим и броим простите делители на всички числа в интервала [2;1000000000].

```
#include<iostream.h>
#include<math.h>
void main()
{ unsigned long o,w[6543],p,k=2,u=7,i;
  bool g,r;
  *w=2;*(w+1)=3;*(w+2)=5;r=false;
  while (u<100000)
   {            p=unsigned long(sqrt(u));g=true;o=1;
                do if (u%*(w+ ++o)==0) {g=false;break;} while(*(w+o)<p);
                if (g) if (k<6542) *(w+ ++k)=u; else break;
                if (r) {r=false;u+=2;} else {r=true;u+=4;}            }
  o=0;
  for(u=2;u<1000000000;u++)
  {p=u;k=unsigned long(sqrt(p));i=0;
   while(p!=1) if (p%*(w+i)==0) {p/=*(w+i);k=long(sqrt(p));o++;}
                               else {if (*(w+i)>k) {o++;break;}i++;}            }
   cout<<o<<endl;
}
```

Резултат от програмата: 4044220040.

III. Да се намерят всички съставни числа n, такива че $n = q_1^{\alpha_1} q_2^{\alpha_2} \ldots q_r^{\alpha_r} = q_1^a + q_2 + \ldots + q_r$ за някое цяло положително a, където $q_1 < q_2 < \ldots < q_r$ са простите делители на n . Условието на проблема е взето от [3].

Идея на метода в общия случай: Търсят се всички прости делители (заедно с кратностите им) на дадено число n (Използват се идеите от II на статията). След което е достатъчно да се провери дали $a = \log_{q_1}\left(n - \left(q_2 + \ldots + q_r\right)\right)$ е цяло.

Програмна реализация: Търсят се всички числа с търсеното свойство в интервала [$2\ ;\ 2^{32}-2$].

```
#include<iostream.h>
#include<math.h>
```

```
void main()
{
  unsigned long x[65],w[6543],p,o,k=2,u=7,i,q,t,y;
  bool g,r,f;
  double lo;
  *w=2;*(w+1)=3;*(w+2)=5;r=false;
  while (u<100000)
   {
              p=unsigned long(sqrt(u));g=true;o=1;
              do if (u%*(w+ ++o)==0) {g=false;break;} while(*(w+o)<p);
              if (g) if (k<6542) *(w+ ++k)=u; else break;
              if (r) {r=false;u+=2;} else {r=true;u+=4;}
   }
  for(u=2;u<4294967295;u++)
  { p=u;k=unsigned long(sqrt(p));i=0;f=true;q=0;
    while(p!=1)
     if (p%*(w+i)==0) {p/=*(w+i);if (f) {x[++q]=*(w+i);x[++q]=1;}
                                  else if (x[q-1]==*(w+i)) x[q]+=1;
                        f=false;k=long(sqrt(p));}
                  else {if (*(w+i)>k) {x[++q]=p;x[++q]=1; break;} i++; f=true;}
            t=0;
            for(i=3;i<q;i+=2)
                        t+=x[i];
            if (x[1]>0&&t>0)
            { lo=log(u-t)/log(x[1]);
                        if (lo-(unsigned long)(log10(u-t)/log10(x[1]))<1e-11)
                        {y=q-1;cout<<u<<"= ";
                        for(i=1;i<y;i+=2) if (x[i+1]==1) cout<<x[i]<<"*";
                                                else cout<<x[i]<<"^"<<x[i+1]<<"*";
                        if (x[i+1]==1) cout<<x[i];else cout<<x[i]<<"^"<<x[i+1];
                        cout<<"= "<<x[1]<<"^"<<lo<<"+";
                        for(i=3;i<y;i+=2) cout<<x[i]<<"+";
                        cout<<x[y]<<endl;
                        }
            }
     }
}
```

Резултат от програмата:

42= 2*3*7= 2^5+3+7

140= 2^2*5*7= 2^7+5+7

290= 2*5*29= 2^8+5+29

618= 2*3*103= 2^9+3+103

2058= 2*3*7^3= 2^11+3+7

6747= 3*13*173= 3^8+13+173

131430= 2*3*5*13*337= 2^17+3+5+13+337

531531= 3^2*7*11*13*59= 3^12+7+11+13+59

2098830= 2*3*5*43*1627= 2^21+3+5+43+1627

5124615= 3*5*341641= 3^14+5+341641

14356161= 3^2*227*7027= 3^15+227+7027

34797196= 2^2*7*1242757= 2^25+7+1242757

40265322= 2*3*6710887= 2^25+3+6710887

67239938= 2*257*130817= 2^26+257+130817

1164192201= 3^2*67*1930667= 3^19+67+1930667

2191309850= 2*5^2*43826197= 2^31+5+43826197

3486789945= 3^2*5*31*499*5009= 3^20+5+31+499+5009.

IV. Дадени са n точки в равнината. Да се намерят координатите на 4 от тях, които могат да са върхове на изпъкнал четириъгълник с максимално лице.

Идея на метода в общия случай: Идеята на алгоритъма е описана в [2].

Програмна реализация: Представената програмна реализация намира едно решение, т.е. една четворка от точки ако задачата има решение. При нужда не представлява трудност координатите на точките да се въвеждат от клавиатурата или да се четат от файл. Тук е реализирано генерирането на случайни стойности за координатите на n -те точки. На n е дадена стойност 100.

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
void main()
 {
            long t[1024][2],Smax,g,n,j,gw1,gw2,gw3,gw4,
                        p1,p2,p3,p4,p5,p6,n1,n2,n3,i1,i2,i3,i4,j1,j2,j3,j4,i11,i21,i31;
            n=100;
 srand((unsigned)time(0));
              for(j=0;j<n;j++)
              {       t[j][0]=(long)((rand()/32768.0-0.5)*RAND_MAX);
           t[j][1]=(long)((rand()/32768.0-0.5)*RAND_MAX);
 }
 Smax=0;n3=n-3;n2=n3+1;n1=n2+1;
 for(i1=0;i1<n3;i1++) {i11=i1+1;
  for(i2=i11;i2<n2;i2++){i21=i2+1;
    p1=t[i2][1]-t[i1][1]; p4=t[i1][0]-t[i2][0];
   for(i3=i21;i3<n1;i3++){i31=i3+1;
     p5=t[i3][0]-t[i1][0]; p6=t[i3][1]-t[i1][1]; gw2=p1*p5+p6*p4;
    for(i4=i31;i4<n;i4++){ p2=t[i4][0]-t[i1][0]; p3=t[i4][1]-t[i1][1];
   gw1=p1*p2+p3*p4; gw3=p6*p2-p3*p5; gw4=gw2+gw3-gw1;
   if (gw1==0||gw2==0||gw3==0) g=0;
      else
   if ((float)gw1/(float)gw2<0&&(float)gw4/(float)gw3<0) g=abs(gw2)+abs(gw1);
      else
   if ((float)gw3/(float)gw2>0&&(float)gw4/(float)gw1>0) g=abs(gw2)+abs(gw3);
      else
   if ((float)gw1/(float)gw3<0&&(float)gw4/(float)gw2<0) g=abs(gw1)+abs(gw3);
      else g=0;
   if (g>Smax) {Smax=g;j1=i1;j2=i2;j3=i3;j4=i4;}
 }}}}
 if (Smax>0) { cout<<"Koordinatite na tochkite, izmejdu tyah zagrajdashti"<<
          " maksimalna plosht ot ravninata mogat da sa: ";
 cout<<"("<<t[j1][0]<<","<<t[j1][1]<<"), ("<<t[j2][0]<<","<<t[j2][1]<<"), ("
             <<t[j3][0]<<","<<t[j3][1]<<"), ("<<t[j4][0]<<","<<t[j4][1]<<")"<<endl;
 cout<<"2*Smax= "<<Smax<<endl;}   else cout<<"Nyama reshenie!"<<endl;
}
```

Резултат от програмата: В общия случай всеки път е различен, тъй като всеки път участват различни точки като входни данни за компютърния алгоритъм.

Предложените подходи за компютърна реализация на някои математически задачи обобщават и оптимизират публикувани алгоритми в [1,2]. Разгледаните проблеми са особено подходящи за решаване с компютърни средства.

Литература

[1] Наков, П., П. Добриков. *Програмиране=++Алгоритми*, С., ТорTeam Co., 2003.

[2] Крушков, Хр., А. Илиев, Практическо ръководство по програмиране на Паскал, Макрос, 2007.

[3] Koninck, J. M. de, A. Mercier, 1001 Problems in Classical Number Theory, American Mathematical Society, 2007.